Open Archive Toulouse Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in: http://oatao.univ-toulouse.fr/
Eprints ID: 9291


Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@inp-toulouse.fr
THE AADL CONSTRAINT ANNEX
Jérome Hugues, ISAE
Serban Georghe, EdgeWater
Agenda

1. Motivation for the Constraints Annex
2. Overview of the current Annex
3. Conclusion and Roadmap
AADL in one slide (!)

Architecture helps you focusing on the actual system

Link to code/model
Workflow with SysML,
Executable models (SCADE, Simulink)
Code (Ada, C, lua, …)

-- Textual AADL

thread Sender_Thread
features
  Data_Source : out event data port Record_Type_Impl;
properties
  Dispatch_Protocol -> Periodic;
  Frequency := 1 Sec;
annex real specification {*
  ** Contract to be enforced
**};
end Sender_Thread;

Non-functional properties

Architectural patterns

SAE INTERNATIONAL
Some examples of AADL tool support

AADL as a backbone, federating multiple activities
• analysis through generation of intermediate models + external tools

Common tool IDE: OSATE2 from SEI (FLOSS)
• AADL core (SEI) + Behavioral (TPT) + Error (SEI) annexes

Non exhaustive list of tools, European-centric (see http://www.aadl.info)
• Integration to a process: with SysML, Simulink, SCADE
• Architectural pattern checks: MILS, ARINC, Ravenscar, Synchronous
• Model checking:
  • Timed/Stochastic/Colored Petri Nets
  • Timed automata et al.: UPPAAL, Versa, TASM
• Scheduling: MAST, Cheddar, CARTS
• Performance evaluation: real-time and network calculus
• Fault analysis: COMPASS, Stochastic Petri Nets, PRISM, FHA
• Simulation: ADeS, Marzhin
• Energy consumption of SoC: OpenPeople project
• Code generation: SystemC, C, Ada, RTSJ, Lustre
• WCET analysis: mapping to Bound-T
Some analysis can be set with a simple DSL: e.g. RMA, using REAL (O. Gilles PhD)

-- Check whether the threads bound to each processor can be scheduled
-- with RMA (cf. Liu, Layland. "Scheduling Algorithms for Multi-programming
-- in hard-Real-Time Environment", JACM, 01/1973)

\begin{verbatim}
theorem RMA
   foreach e in Processor_Set do
      Proc_Set(e) := \{ x in Process_Set | Is_Bound_To (x, e) \};
      Threads := \{ x in Thread_Set | Is_Subcomponent_Of (x, Proc_Set) \};
      check (sum (get_property_value (Threads, "RTOS_properties::Utilization")) <=
      (Cardinal (Threads) * (2 ** (1 / Cardinal (Threads))) -1));
   end RMA;
\end{verbatim}

Some others require mapping to another model, follow a regular pattern

1. Define a subset of AADL (e.g. ARINC, MILS)
2. Define its semantics (valid combination of elements, properties)
3. Define a mapping to an external tool (e.g. checker)

Pros: works well, easy to prototype in OSATE2/Eclipse
Cons: subset and semantics encoded in mapping tool, no consensus or interoperability on subsets, e.g. for schedulability
Example#2: PERSEUS supersonic rocket

Analysis of rocket kinematics performance

- Mass of each element of the rocket
- Total rocket mass
- Trajectory computation
- Engines thrust
- Maximal acceleration
- Maximal compression loads during flight
- Compression loads withstood
- Mass of the section above current section
- Does this section withstand compression loads?

- Drag coefficient
- Aerodynamics areas

- Constraints theorems
- Performances theorems
- Data from implementation
Agenda

1. Motivation for the Constraints Annex
2. Overview of the current Annex
3. Conclusion and Roadmap
1. **Structural Assertions on Models**
   
a. Semantics of project specific Property sets and values
   • Consistency and completeness of Property Associations
   • System instance specific global constraints on Property types

b. Modeling assumptions of viewpoint specific Property sets/values
   • Capture assumptions of viewpoint specific abstractions used by modeling tools
   • functional, real-time, safety, security, etc…

c. Enforcement of AADL modeling subsets and option choices

2. **Behavior (Temporal) Assertions**
   
a. Component centric Assume/Guarantee specification contracts (Contract Based Development)
   • Black box – defined using only component interface features, consistent with the AADL Component Type declarations and the component extension/refinement mechanisms

b. System level safety and liveness temporal formal properties verification
   • Black box - Assume/Guarantee specification contracts for top System component
   • White box - Assumes a deployable system instance and global scope
### Constraint Annex Grammar Layers

<table>
<thead>
<tr>
<th>Constraint Annex sublanguages organized as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A set of Grammar layers, built on top of Property Expressions (see picture):</td>
</tr>
<tr>
<td>1) Property expressions and Constants (already exists)</td>
</tr>
<tr>
<td>2) New grammar rules (or built-in functions) for accessing Features data details of Component Types/Subcomponents compositions/ Feature groups</td>
</tr>
<tr>
<td>3) Relational and Boolean expressions grammar rules</td>
</tr>
<tr>
<td>4) Pre-defined (built-in) model traversal access functions</td>
</tr>
<tr>
<td>5) Reusable static scalar functions / predicates</td>
</tr>
<tr>
<td>  - Constraint Annex Library declarations</td>
</tr>
<tr>
<td>   - Package level declarations</td>
</tr>
<tr>
<td>   - Can be used globally</td>
</tr>
<tr>
<td>  - Constraint Annex subclause declarations for</td>
</tr>
<tr>
<td>   - Component Types, Implementations, Feature Groups and their extensions</td>
</tr>
<tr>
<td>   - Can be used locally</td>
</tr>
<tr>
<td>6) Set building operators and reusable set building functions grammar</td>
</tr>
<tr>
<td>  - Annex Library (global use) or Annex subclause in components</td>
</tr>
<tr>
<td>7) State Sequence expressions grammar</td>
</tr>
<tr>
<td>  - PSL SERE subset referencing features in Feature Groups, Component Type (interface), connected subcomponent subsets</td>
</tr>
</tbody>
</table>

#### (1) AADL Property Expressions and Constants

#### (2) Feature Data Reference grammar (or built-in functions) (Component Types, Subcomponents, Feature groups)

#### (3) Relational and Boolean Expressions grammar rules

#### (4) Built-in Model Traversal Access Functions

#### (5) Static scalar Functions/Predicates Declaration and Use grammar rules

#### (6) Set building sublanguage grammar (Lute/Real)

#### (7) State Sequence Expressions sublanguage grammar (PSL subset)

#### (8) Theorems in Annex subclause (Lute/Real)

#### (9) Temporal Formulas in Annex subclause (PSL subset)

#### (10) Assumptions, Guarantees, Composition Facts In Annex subclause

#### (11) Component Contract Viewpoints Declaration and Use grammar

#### (12) Instance Contract Viewpoints Declaration and Use grammar
Component and Instance Contracts

<table>
<thead>
<tr>
<th>(12) Component Contract Viewpoints Declaration and Use grammar</th>
<th>(9) Instance Contract Viewpoints Declaration and Use grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>(11) Assumptions, Guarantees, Composition Facts</td>
<td></td>
</tr>
<tr>
<td>In Annex subclause</td>
<td>(8) Theorems</td>
</tr>
<tr>
<td>in Annex subclause</td>
<td>in Annex subclause</td>
</tr>
<tr>
<td>(10) Temporal Formulas</td>
<td></td>
</tr>
<tr>
<td>in Annex subclause</td>
<td>(Lute/Real)</td>
</tr>
<tr>
<td>(7) State Sequence Expressions sublanguage grammar (PSL subset)</td>
<td></td>
</tr>
<tr>
<td>(6) Set building sublanguage grammar (Lute/Real)</td>
<td></td>
</tr>
<tr>
<td>(5) Built-in Model Traversal Access Functions</td>
<td></td>
</tr>
<tr>
<td>(4) Static scalar Functions/Predicates Declaration and Use grammar rules</td>
<td></td>
</tr>
<tr>
<td>(3) Relational and Boolean Expressions grammar rules</td>
<td></td>
</tr>
<tr>
<td>(2) Feature Data Reference grammar (or built-in functions)</td>
<td></td>
</tr>
<tr>
<td>(Component Types, Subcomponents, Feature groups)</td>
<td></td>
</tr>
<tr>
<td>(1) AADL Property Expressions and Constants</td>
<td></td>
</tr>
</tbody>
</table>

8) Theorems: based on Lute/Real languages

9) Grammar for grouping Theorems into standard or custom defined **Instance Contract Viewpoints**

10) Subset of PSL Foundation Logic (LTL) temporal operators on Boolean and State Sequence Expressions

11) **Assumptions** specify *temporal invariants* conditioned by component outputs, **Guarantees** specify temporal invariants conditioned by component inputs. **Composition facts** specify temporal invariants on sets of subcomponents inputs and outputs

12) Grammar for grouping Assumptions, Guarantees, Composition Facts and into standard or custom defined Component Contract Viewpoints

***) See A. Pnueli & A. Zachs – PSL Model Checking and Run Time Verification via Testers
Black Box and White Box Contracts

**Component Contracts**

1. AADL Property Expressions and Constants
2. Feature Data Reference grammar (or built-in functions)
3. Relational and Boolean Expressions grammar rules
4. Static scalar Functions/Predicates Declaration and Use grammar rules
5. Built-in Model Traversal Access Functions
6. Set building sublanguage grammar (Lute/Real)
7. State Sequence Expressions sublanguage grammar (PSL subset)
8. Temporal Formulas in Annex subclause (PSL subset)
9. Instance Contract Viewpoints Declaration and Use grammar
10. Assumptions, Guarantees, Composition Facts In Annex subclause
11. Instance Contract Viewpoints Declaration and Use grammar
12. Component Contract Viewpoints Declaration and Use grammar

**Instance Contracts**

- Theorems in Annex subclause (Lute/Real)
- Assumptions, Guarantees, Composition Facts in Annex subclause

**Black Box (Compositional) Specification Contracts**
- Component Centric, must work with component “extends” and “refines” constructs

**White Box Structural Contracts**
- Instance Model Centric
package LanguageSubsets
public
  -- A Constraint Annex library to declare a few contract viewpoints
  annex Constraint_Annex {**

  viewpoint AADL_light subset contract { -- Structural Assertion script for AADL light constraints
    }

  Ravenscar subset contract applies to system implementation {-- Structural Assertion script for Ravenscar
    }

  **}
end LanguageSubsets;

package CompliantPkg
public
  with LanguageSubsets;
  annex Constraint_Annex {**
    enforce subset contract AADL_light
      subset contract Ravenscar in binding -- some binding
    **}
end CompliantPkg;
-- This example shows the structure of a Constraint Annex library and a Constraint Annex sub-clause

package ContractViewpoints
public
-- A Constraint Annex library to declare a few contract viewpoints and reusable
-- utility functions
annex Constraints_Annex {**
viewpoint PalsChecks structural contract;
-- Other contract viewpoints can be declared here
viewpoint OtherViewpoint …;
functions ...
Processor_Set (root : T_Subcomponent) := set of T_Processor from root;
Thread_Set := set of T_Thread from self;
Connection_Set = set of T_Connection from self ;
Max_Thread_Jitter(Threads : set of T_Thread) returns aadlreal :=
Max({Property(p, "Clock_Jitter") for p in Processor_Set(self) |
Cardinal({t in Threads | Is_Bound_To(t, p)}) > 0});
Connections_Among(Set : set of T_Thread) returns set of T_Connection :=
{c in Connection_Set | Member(Owner(Source(c)), Set) and
Member(Owner(Destination(c)), Set)};
}
**}
end ContractViewpoints;

- A contract viewpoint declared in a Constraint Annex library
- Will be referenced in multiple classifier annex sub-clauses
- Contract enforced by each classifier through a constraint script
- A parametrized function expression definition
- It uses an iterative set mapping expression over Processor_Set
and pre-defined utility functions: Max, Cardinal, Is_Bound_To
- Two different ways of declaring containment set functions
- When called without a parameter, Process_Set will assume as default
  parameter the current classifier
- Another parametrized function expression definition
- It uses an iteration over Connection_Set and
  pre-defined utility functions: Member, Owner, Source, Destination
Agenda

1. Motivation for the Constraints Annex
2. Overview of the current Annex
3. Conclusion and Roadmap
AADL Constraints Annex relies on many previous experiments around AADL

- REAL: O. Gilles PhD to enforce structural constraints on AADL models
- Lute: extensions to REAL by Rockwell Collins
- RDAL: attaching requirements and traceability functions to AADL entities

Annex main goal is to turn those positive feedbacks into a concrete standards to reduce the gap between model and V&V activities

- Key requirement: language should be used by non-expert in MDE, but system engineers

Issues and roadmap

- Implementation of a prototype (Q3 2014)
- Adding temporal constraints, borrowing concepts from IEEE PSL standard
- Connection with external tools: Constraints Annex as set of accessors to bridge to external tools
- Testing concepts on existing library of constraints
  - AADL subsets: IMA, ARINC653, Ravenscar, Time-Triggered, Synchronous
  - Project-specific analysis: PERSEUS rocket, UAVs, etc.

Join SAE AS2-C for more details and discussion!